Steven Lee, Allister Liu, Amy Leong
Operating Systems Problem Set 7

1A) A user mode process attempts to make a system call with the system call number set to -1. Refer to the X86-32 entry.S code in the lecture notes and explain what happens.

The system call number -1 is passed into the eax register to prepare for the system call. However, a system call number of -1 is treated like an error. When a user mode process attempts to make an invalid system call, it will return the value -ENOSYS via the %eax register. In other words, when this system call fails, it will set the global variable errno to -%eax (-ENOSYS), and return -1.

1B) After a system call is completed and the kernel returns control to the user-level process which invoked the system call, how does the user/supervisor privilege level get reset to user mode?

When resetting to user mode after returning from a system call, all registers have to be restored and then iret and sysexit are executed to restore the old instruction and stack pointers at user-level. Specifically, the return value from the system call is passed in the %eax register where the register will be popped when returning back to user mode. Then, the syscall_exit interrupts are masked temporarily (to protect a critical region). If all flags are clear, RESTORE_REGS pops all the registers from SAVE_REGS and the extra stacked copy of eax is discarded. When iret executes, the hardware restores the eip, cs, eflags, esp and ss registers. This resets to user mode by having the stack pointer on the user's stack and all the registers are restored to how it was before, except with the eax register containing the system call return value.

1C) Process 999 has made a system call which encountered a blocking condition: waiting for disk I/O. Assume a single-CPU system for simplicity. While this is taking place, the CPU has moved on and is running process 123. Process 123 makes a system call. During the handling of that system call, the disk I/O interrupt arrives. The completion of disk I/O unblocks pid 999, which is now in the "READY" state. Describe what happens next in (i) a fully pre-emptive Linux kernel (ii) a non pre-emptive Linux kernel

i) For a fully pre-emptive Linux kernel, upon return from the interrupt handler to the kernel system call handler, the NEED_RESCHED flag is noticed and then a context switch would take place where process 123 would be suspended (At the moment when the interrupt handler arrived) and process 999 gets the CPU. Afterward, process 123 would resume.

ii) For a non-preemptive Linux kernel, the interrupt handler completes and controls resumes in kernel code, continuing the syscall in process 123. Once the system call in process 123 completes, pre-emption happens and the CPU does not return to the user-mode process directly. It would check the NEED_RESCHED flag and if it set, then the kernel would pick a process from all the processes to determine which one is fit to run next. If a different process is selected, a context switch happens.